Département Informatique Année Universitaire 2022-2023

Chapitre 4

Lossy Compression

Plan

- **Image Compression: Key Ingredients**
- **JPEG: Joint Photographic Experts Group**

- **Image Compression: Key Ingredients**

Key ingredients

- compressing at block level
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Please meet ...

Compressing at pixel level

- reduce number bits per pixel
- equivalent to coarser quantization
- in the limit, 1bpp

Compressing at block level

- divide the image in blocks
- code the average value with 8 bits
- 3×3 blocks at 8 bits per block gives less than 1bpp

Compressing at block level

- exploit the local spatial correlation
- compress remote regions independently

Transform coding

Ideally, we would like a transform that:

- captures the important features of an image block in a few coefficients
- is efficient to compute

- answer: the Discrete Cosine Transform

We'll use a grayscale image

Suppose this is the image that our camera "sees" through its lens.

But the same approach would apply to colored images.

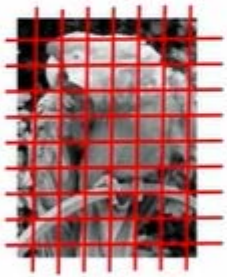
Divide the image into pixels.

Suppose square in the grid represents 1 ***pixel***.

- Here we have a 7 x 9 grid, 63 pixels.

Represent each pixel by a number

Suppose we represent each pixel in the photo by its brightness, with 255 representing white and 0 representing black..



255 white, maximum brightness

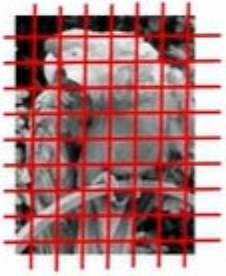
100 light grey

50 dark grey

0 black, minimum brightness

Numbers represent light intensity

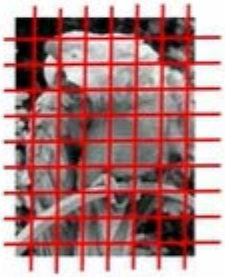
Then we can represent the image completely by these numbers.



00	25	50	48	47	42	40
10	100	240	220	210	185	50
08	120	190	220	205	201	190
85	150	148	147	152	163	165
135	146	142	147	167	163	161
84	143	152	147	167	160	161
89	135	162	142	157	150	151
101	121	151	140	152	138	131
101	115	136	161	124	89	42

Of course, it's all bits

Of course, in the computer's memory, these decimal numbers are all ***bits***, maybe 8 bits per pixel for a grayscale image.



```
00  25   50  48   47   42   40
10 100  240 220  210 185   50
08 120  190 220  205 201  190
85 150  148 147  152 163  165
135 146 142 147  167 163  161
84 143  152 147  167 160  161
89 135  162 142  157 150  151
101 121 151 140  152 138  131
101 115 136 161  124  89   42
```

```
00000000000011001001100100011000...
00001010101010100100101001001010...
00001000101010001010101010100100...
01010001010010010010100101010010..
01101010100010101010101000000010...
01100101010001010100010101010001...
01000100010101010101000101010000...
01001010010101001000100000100010...
01101010010001110001101000010100....
```

Consider a row of 7 pixels

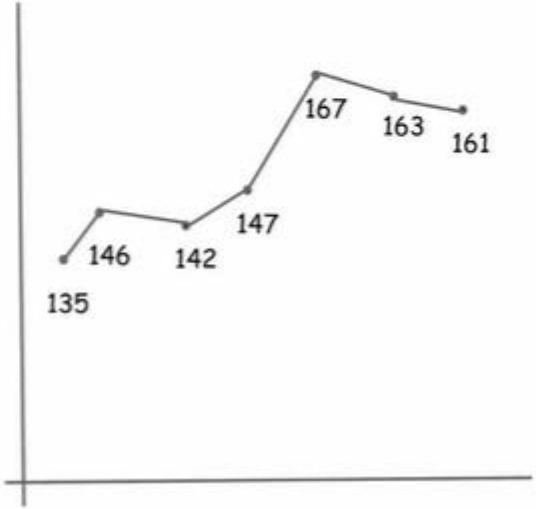
Now suppose we look at 1 row of the image

00	25	50	48	47	42	40
10	100	240	220	210	185	50
08	120	190	220	205	201	190
85	150	148	147	152	163	165
135	146	142	147	167	163	161
84	143	152	147	167	160	161
89	135	162	142	157	150	151
101	121	151	140	152	138	131
101	115	136	161	124	89	42

And graph the points

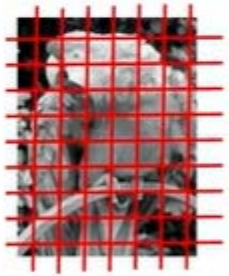
And graph the points in that row.

00	25	50	48	47	42	40
10	100	240	220	210	185	50
08	120	190	220	205	201	190
85	150	148	147	152	163	165
135	146	142	147	167	163	161
84	143	152	147	167	160	161
89	135	162	142	157	150	151
101	121	151	140	152	138	131
101	115	136	161	124	89	42

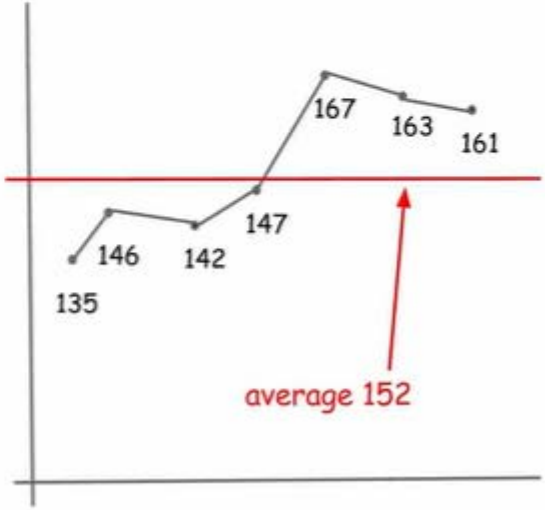


Average the points

And take the average of those points (152).

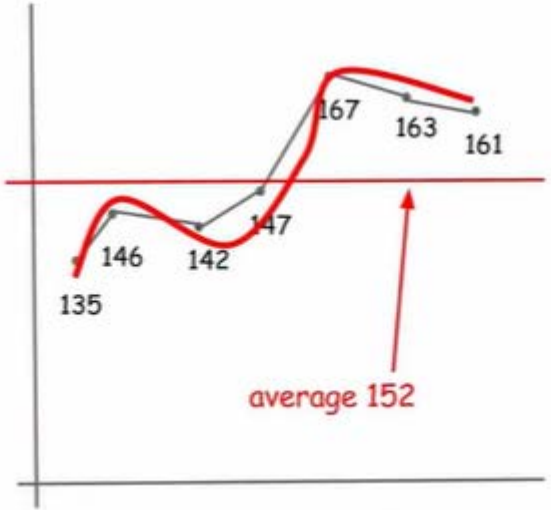


00	25	50	48	47	42	40
10	100	240	220	210	185	50
08	120	190	220	205	201	190
85	150	148	147	152	163	165
135	146	142	147	167	163	161
84	143	152	147	167	160	161
89	135	162	142	157	150	151
101	121	151	140	152	138	131
101	115	136	161	124	89	42



Approximate the points

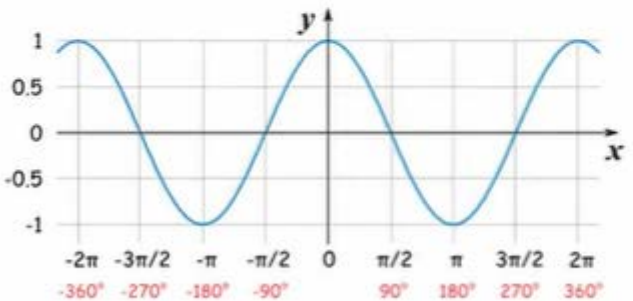
Mathematicians have figured that we can ***approximate*** that set of points with a ***cosine*** curve that looks something like this.



152 + 10 cos(X) - 5 sin(x)

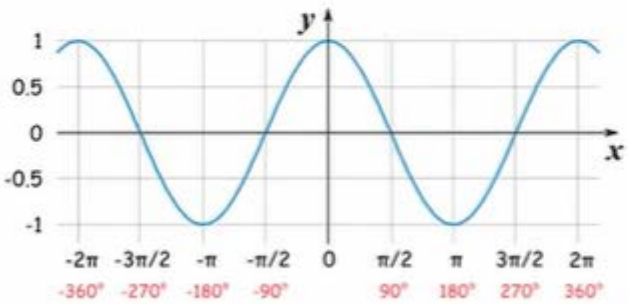
Cosine Function

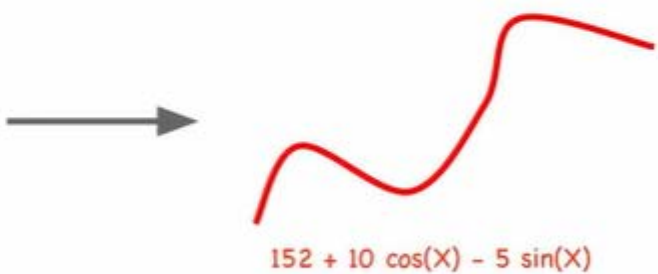
Our curve is just a ***transformation*** of the cosine curve that you learned in high school, hence the name ***discrete cosine transform***.



Cosine Function

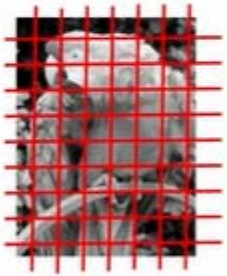
Our curve is just a ***transformation*** of the cosine curve that you learned in high school, hence the name ***discrete cosine transform***.

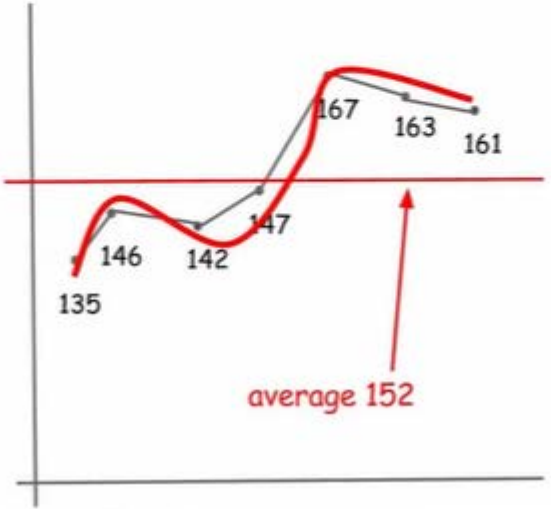




Approximate the points

Mathematicians have figured that we can ***approximate*** that set of points with a ***cosine*** curve that looks something like this.

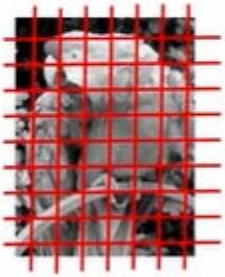




152 + 10 cos(X) - 5 sin(x)

Compression: approximate the points

So instead of representing that line of pixels with 7 numbers we can now represent it with 3 numbers, the coefficients of the cosine curve. .



00	25	50	48	47	42	40
10	100	240	220	210	185	50
08	120	190	220	205	201	190
85	150	148	147	152	163	165
135	146	142	147	167	163	161
84	143	152	147	167	160	161
89	135	162	142	157	150	151
101	121	151	140	152	138	131
101	115	136	161	124	89	42

135 146 142 147 167 163 161

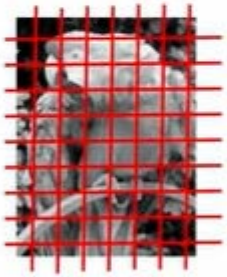
Compression

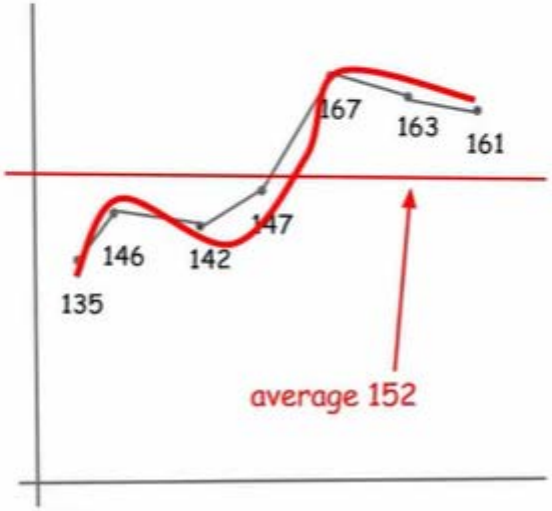
152 10 5

152 + 10 cos(X) - 5 sin(X)

Lossy: Can't get the points back

This is ***lossy compression*** because we can't recover the original numbers from the cosine curve. That information is lost.





152 + 10 cos(X) - 5 sin(x)

135 146 142 147 167 163 161

2D-DCT

$$C[k_1, k_2] = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x[n_1, n_2] \cos\left[\frac{\pi}{N}\left(n_1 + \frac{1}{2}\right) k_1\right] \cos\left[\frac{\pi}{N}\left(n_2 + \frac{1}{2}\right) k_2\right]$$

$N = 8$

$N = 8$

- **JPEG: Joint Photographic Experts Group**

Key ingredients

- split image into 8×8 non-overlapping blocks
- compute the DCT of each block
- quantize DCT coefficients according to psycovisually-tuned tables
- run-length encoding and Huffman coding

Introduction

- JPEG : Joint Photographic Experts Group
  - Original name
    - The committee of the International Organization for Standardization (ISO)
  - The first international static image compression standard Published in 1992： ISO 10918-1
- Because of its pleasing properties, JPEG gained great success only several years after published
  - Almost 80 percents of images on web are compressed by the JPEG standards

Introduction

- JPEG is a lossy image compression method. It employs a transform coding method using the DCT (Discrete Cosine Transform).

- An image is a function of i and j (or conventionally x and y) in the spatial domain. The 2D DCT is used as one step in JPEG in order to yield a frequency response which is a function F(u, v) in the spatial frequency domain, indexed by two integers u and v.

Observations for JPEG Image Compression

- The effectiveness of the DCT transform coding method in JPEG relies on 3 major observations:

Observation 1: Useful image contents change relatively slowly across the image, i.e., it is unusual for intensity values to vary widely several times in a small area, for example, within an 8×8 image block.

- much of the information in an image is repeated, hence “spatial redundancy”.

Observations for JPEG Image Compression

Observation 2: Psychophysical experiments suggest that humans are much less likely to notice the loss of very high spatial frequency components than the loss of lower frequency components.

- the spatial redundancy can be reduced by largely reducing the high spatial frequency contents.

Observation 3: Visual acuity (accuracy in distinguishing closely spaced lines) is much greater for gray (“black and white”) than for color.

- chroma subsampling (4:2:0) is used in JPEG.

1.1 Main Steps in JPEG Image Compression

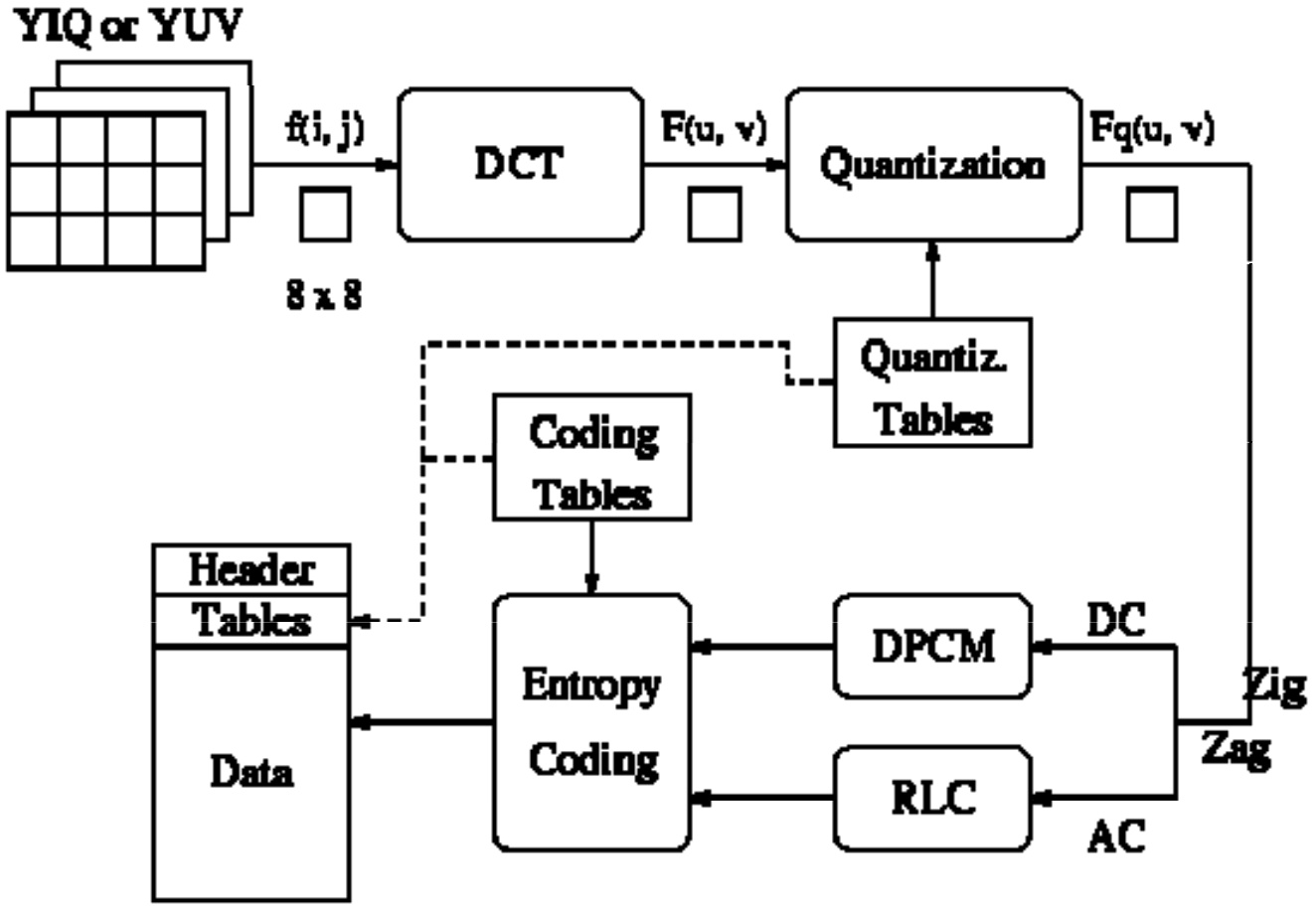
(1) Transform RGB to YIQ or YUV and subsample color

(2) Perform DCT on image blocks
(3) Apply Quantization

(4) Zigzag Ordering
(5) DPCM on DC coefficients
(6) RLE on AC coefficients

(7) Perform entropy coding

1.1 Main Steps in JPEG Image Compression

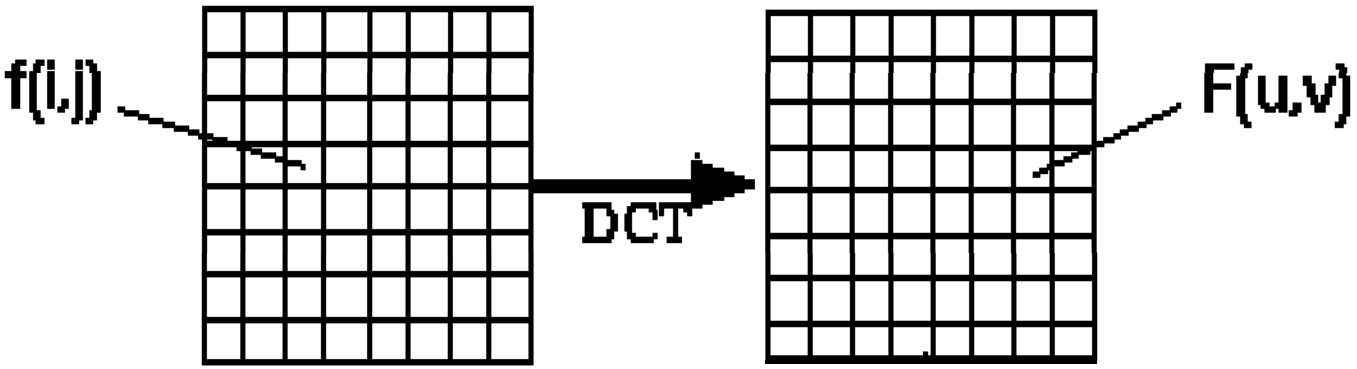


Block diagram for JPEG encoder

1.1 Main steps: DCT

DCT (Discrete Cosine Transformation)

Each image is divided into 8 × 8 blocks. The 2D DCT is applied to each block image $f(i, j)$, with output being the DCT coefficients $F(u, v)$ for each block.



1.1 Main steps: DCT

- Why the block size is 8×8?
  - Compromise between accuracy and computation

- Removing blocking artifacts is an important concern of researcher

- Using blocks, however, has the effect of isolating each block from its neighboring context. This is why JPEG images look choppy (“blocky”) when a high *compression ratio* is specified by the user.

Quantization

- Why? -- To reduce number of bits per sample

  $F'(u,v) = \text{round}(F(u,v)/q(u,v))$
- Example: 101101 = 45 (6 bits).
  Truncate to 4 bits: 1011 = 11. (Compare 11 x 4 =44 against 45)
  Truncate to 3 bits: 101 = 5. (Compare 8 x 5 =40 against 45)
  Note, that the more bits we truncate the more precision we lose
- Quantization error is the main source of the Lossy Compression.
- **Uniform Quantization:**
  - q(u,v) is a constant.
- **Non-uniform Quantization -- Quantization Tables**
  - Eye is most sensitive to low frequencies (upper left corner in frequency matrix), less sensitive to high frequencies (lower right corner)
  - Custom quantization tables can be put in image/scan header.
  - JPEG Standard defines two default quantization tables, one each for luminance and chrominance.

1.1 Main steps: Quantization

$$\hat{F}(u,v) = round\left(\frac{F(u,v)}{Q(u,v)}\right) \qquad (9.1)$$

- $F(u, v)$ represents a DCT coefficient, $Q(u, v)$ is a "quantization matrix" entry, and $\hat{F}(u,v)$ represents the *quantized DCT coefficients* which JPEG will use in the succeeding entropy coding.
  - The quantization step is the main source for loss in JPEG compression.
  - The entries of $Q(u, v)$ tend to have larger values towards the lower right corner. This aims to introduce more loss at the higher spatial frequencies — a practice supported by Observations 1 and 2.
  - Table 9.1 and 9.2 show the default $Q(u, v)$ values obtained from psychophysical studies with the goal of maximizing the compression ratio while minimizing perceptual losses in JPEG images.

1.1 Main steps: Quantization

Table 9.1 The Luminance Quantization Table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table 9.2 The Chrominance Quantization Table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

1.1 Main steps: Quantization

An 8 × 8 block from the Y image of 'Lena'

200	202	189	188	189	175	175	175
200	203	198	188	189	182	178	175
203	200	200	195	200	187	185	175
200	200	200	200	197	187	187	187
200	205	200	200	195	188	187	175
200	200	200	200	200	190	187	175
205	200	199	200	191	187	187	175
210	200	200	200	188	185	187	186

$f(i, j)$

515	65	-12	4	1	2	-8	5
-16	3	2	0	0	-11	-2	3
-12	6	11	-1	3	0	1	-2
-8	3	-4	2	-2	-3	-5	-2
0	-2	7	-5	4	0	-1	-4
0	-3	-1	0	4	1	-1	0
3	-2	-3	3	3	-1	-1	3
-2	5	-2	4	-2	2	-3	0

$F(u, v)$

Fig. 9.2: JPEG compression for a smooth image block.

1.1 Main steps: Quantization

32	6	−1	0	0	0	0	0
−1	0	0	0	0	0	0	0
−1	0	1	0	0	0	0	0
−1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$\hat{F}(u, v)$

512	66	−10	0	0	0	0	0
−12	0	0	0	0	0	0	0
−14	0	16	0	0	0	0	0
−14	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$\tilde{F}(u, v)$

199	196	191	186	182	178	177	176
201	199	196	192	188	183	180	178
203	203	202	200	195	189	183	180
202	203	204	203	198	191	183	179
200	201	202	201	196	189	182	177
200	200	199	197	192	186	181	177
204	202	199	195	190	186	183	181
207	204	200	194	190	187	185	184

$\tilde{f}(i, j)$

1	6	−2	2	7	−3	−2	−1
−1	4	2	−4	1	−1	−2	−3
0	−3	−2	−5	5	−2	2	−5
−2	−3	−4	−3	−1	−4	4	8
0	4	−2	−1	−1	−1	5	−2
0	0	1	3	8	4	6	−2
1	−2	0	5	1	1	4	−6
3	−4	0	6	−2	−2	2	2

$(i, j) = f(i, j) - \tilde{f}(i, j)$

Fig. 9.2 (cont'd): JPEG compression for a smooth image block.

1.1 Main steps: Quantization

Another 8 × 8 block from the Y image of 'Lena'

70	70	100	70	87	87	150	187
85	100	96	79	87	154	87	113
100	85	116	79	70	87	86	196
136	69	87	200	79	71	117	96
161	70	87	200	103	71	96	113
161	123	147	133	113	113	85	161
146	147	175	100	103	103	163	187
156	146	189	70	113	161	163	197

f(i, j)

-80	-40	89	-73	44	32	53	-3
-135	-59	-26	6	14	-3	-13	-28
47	-76	66	-3	-108	-78	33	59
-2	10	-18	0	33	11	-21	1
-1	-9	-22	8	32	65	-36	-1
5	-20	28	-46	3	24	-30	24
6	-20	37	-28	12	-35	33	17
-5	-23	33	-30	17	-5	-4	20

F(u, v)

Fig. 9.2: JPEG compression for a smooth image block.

1.1 Main steps: Quantization

−5	−4	9	−5	2	1	1	0
−11	−5	−2	0	1	0	0	−1
3	−6	4	0	−3	−1	0	1
0	1	−1	0	1	0	0	0
0	0	−1	0	0	1	0	0
0	−1	1	−1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$\hat{F}(u, v)$

−80	−44	90	−80	48	40	51	0
−132	−60	−28	0	26	0	0	−55
42	−78	64	0	−120	−57	0	56
0	17	−22	0	51	0	0	0
0	0	−37	0	0	109	0	0
0	−35	55	−64	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$\tilde{F}(u, v)$

70	60	106	94	62	103	146	176
85	101	85	75	102	127	93	144
98	99	92	102	74	98	89	167
132	53	111	180	55	70	106	145
173	57	114	207	111	89	84	90
164	123	131	135	133	92	85	162
141	159	169	73	106	101	149	224
150	141	195	79	107	147	210	153

$\tilde{f}(i, j)$

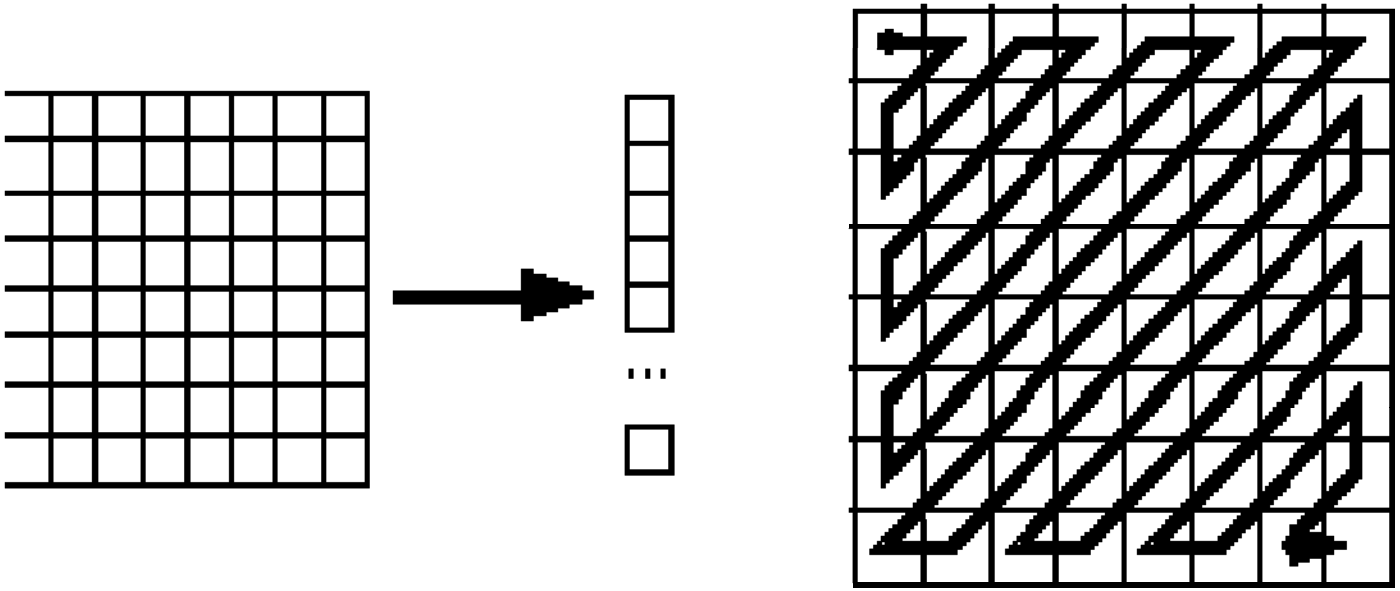
0	10	−6	−24	25	−16	4	11
0	−1	11	4	−15	27	−6	−31
2	−14	24	−23	−4	−11	−3	29
4	16	−24	20	24	1	11	−49
−12	13	−27	−7	−8	−18	12	23
−3	0	16	−2	−20	21	0	−1
5	−12	6	27	−3	−2	14	−37
6	5	−6	−9	6	14	−47	44

$(i, j) = f(i, j) - \tilde{f}(i, j)$

Fig. 9.3 (cont'd): JPEG compression for a textured image block.

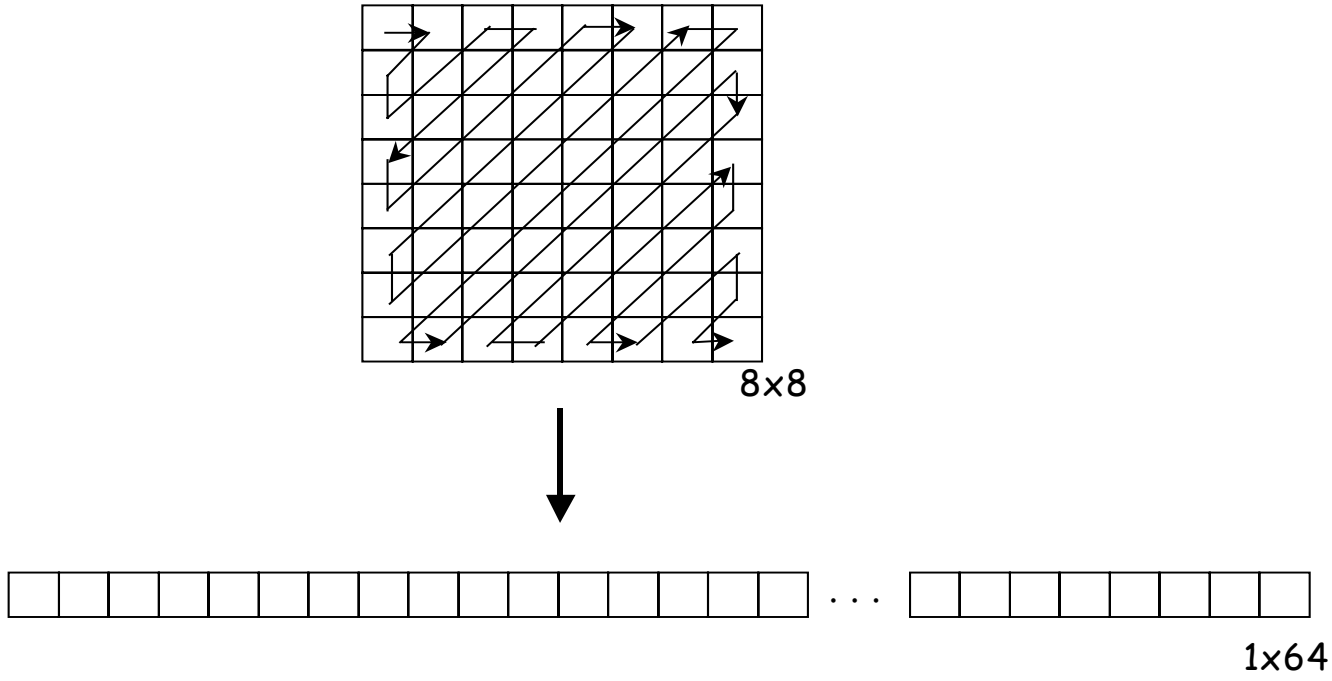
1.1 Main steps: Zigzag Scan

- Turns the 8×8 matrix into a 64 vector
  - Lower frequency components are at the front part of the vector
  - The higher frequency component at the rear part



Zig-Zag Scan

- Why? -- to group low frequency coefficients in top of vector and high frequency coefficients at the bottom
- Maps 8 x 8 matrix to a 1 x 64 vector



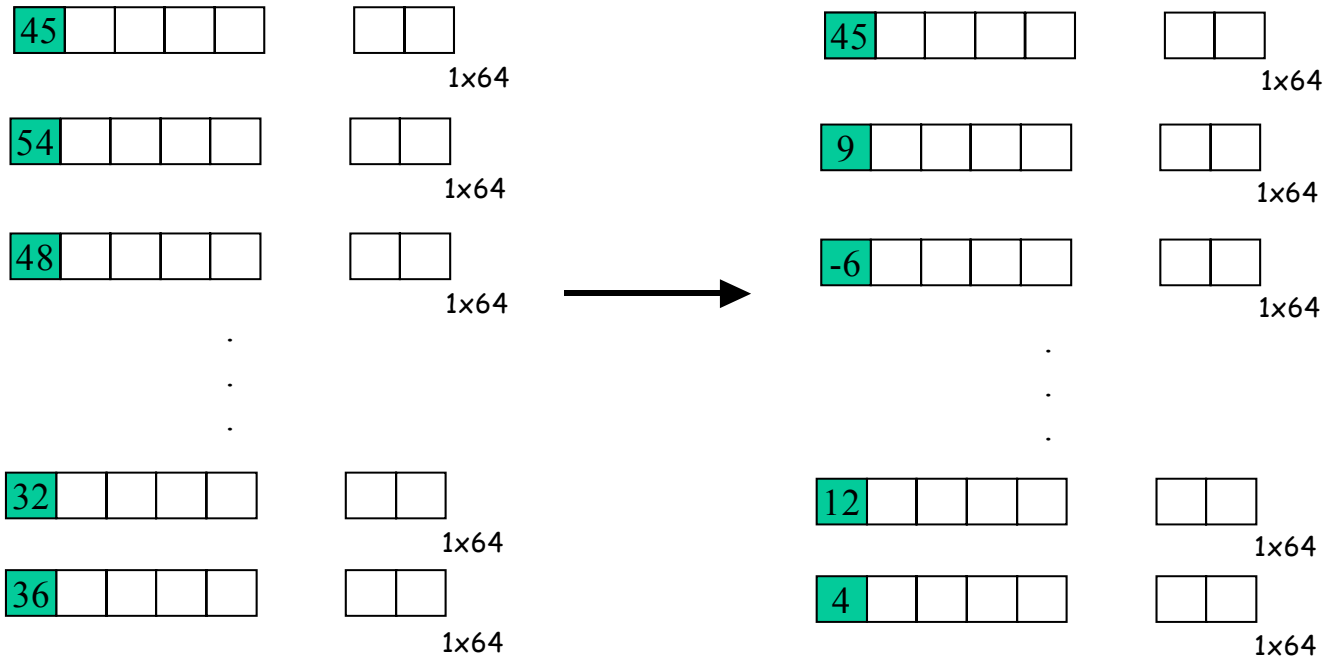
Example

$$\begin{bmatrix} 100 & -60 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 13 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

100, -60, 0, 0, 0, 0, 6, 0, 0, 0, 13, 0, 0, 0, 0, 0, 0, 0, 0, 13, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

DPCM on DC Components

- The DC component value in each 8x8 block is large and varies across blocks, but is often close to that in the previous block.
- Differential Pulse Code Modulation (DPCM): Encode the difference between the current and previous 8x8 block. Remember, smaller number -> fewer bits



Entropy Coding: DC Components

- DC components are differentially coded as (**SIZE**,**Value**)
  - The code for a **Value** is derived from the following table

SIZE	Value	Code
0	0	---
1	-1,1	0,1
2	-3, -2, 2,3	00,01,10,11
3	-7,…, -4, 4,…, 7	000,…, 011, 100,…111
4	-15,…, -8, 8,…, 15	0000,…, 0111, 1000,…, 1111
.		.
.		.
11	-2047,…, -1024, 1024,… 2047	…

Size_and_Value Table

Entropy Coding: DC Components (Contd..)

- DC components are differentially coded as (**SIZE**,**Value**)
  - The code for a **SIZE** is derived from the following table

SIZE	Code Length	Code
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Huffman Table for DC component SIZE field

Example: If a DC component is 40 and the previous DC component is 48. The difference is -8. Therefore it is coded as:

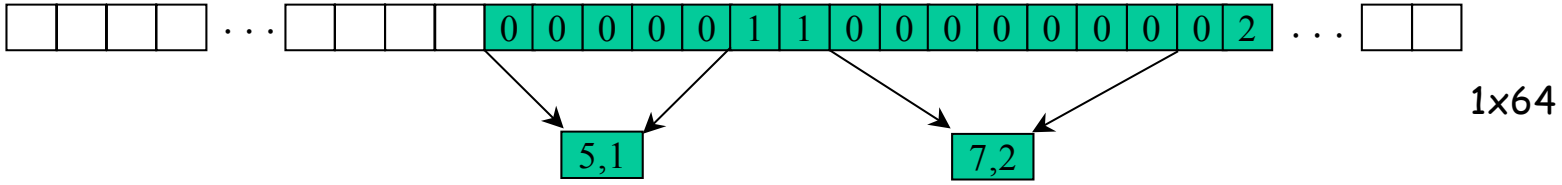
1010111

0111: The value for representing -8 (see size and value table in previous slide)

101: The size from the same table reads 4. The corresponding code from the table at left is 101.

RLE on AC Components

- The 1x64 vectors have a lot of zeros in them, more so towards the end of the vector.
  - Higher up entries in the vector capture higher frequency (DCT) components which tend to be capture less of the content.
  - Could have been as a result of using a quantization table
- Encode a series of 0s as a (*skip*,*value*) pair, where *skip* is the number of zeros and *value* is the next non-zero component.
  - Send (0,0) as end-of-block sentinel value.



Entropy Coding: AC Components

- AC components (range –1023..1023) are coded as (S1,S2 pairs):
  - **S1: (RunLength/SIZE)**
    - **RunLength**: The length of the consecutive zero values [0..15]
    - **SIZE**: The number of bits needed to code the *next* nonzero AC component's value. [0-A]
    - (0,0) is the End_Of_Block for the 8x8 block.
    - **S1** is Huffman coded (see AC code table below)
  - **S2: (Value)**
    - **Value**: Is the value of the AC component.(refer to size_and_value table)

Run/ SIZE	Code Length	Code
0/0	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011

Run/ SIZE	Code Length	Code
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
… 15/A	More	Such rows

Partial Huffman Table for AC Run/Size Pairs

Entropy Coding: Example

40	12	0	0	0	0	0	0
10	-7	-4	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Example: Consider encoding the AC components by arranging them in a zig-zag order -> 12,10, 1, -7 2 0s, -4, 56 zeros

12: read as zero 0s,12: (0/4)12 → 10111100

1011: The code for (0/4 from AC code table)

1100: The code for 12 from the size_and_Value table.

10: (0/4)10 → 10111010

1: (0/1)1 → 001

-7: (0/3)-7 → 100000

2 0s, -4: (2/3)-4 → 1111110111011

1111110111: The 10-bit code for 2/3

011: representation of -4 from size_and_Value table.

56 0s: (0,0) → 1010 (Rest of the components are zeros therefore we simply put the EOB to signify this fact)

Note: For DC component see slide 13